A Parallel Software-Platform for Solving Problems of Partial Differential Equations using Unstructured Grids and Adaptive Multigrid Methods

Peter Bastian, Klaus Johannsen, Stefan Lang,
Sandra Nägele, Christian Wieners, Volker Reichenberger,
Gabriel Wittum, Christian Wrobel

published in

NIC Symposium 2001, Proceedings,
Horst Rollnik, Dietrich Wolf (Editor),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **9**, ISBN 3-00-009055-X, pp. 503-514, 2002.

A Parallel Software-Platform for Solving Problems of Partial Differential Equations using Unstructured Grids and Adaptive Multigrid Methods

Peter Bastian, Klaus Johannsen, Stefan Lang, Sandra Nägele, Christian Wieners, Volker Reichenberger, Gabriel Wittum, and Christian Wrobel

IWR/Technical Simulation, University of Heidelberg
Im Neuenheimer Feld 368, 69120 Heidelberg, Germany

The program package *UG* provides a software platform for discretizing and solving partial differential equations. It supports high level numerical methods for unstructured grids on massively parallel computers. Various applications of complex up to real-world problems have been realized, like Navier-Stokes problem with turbulence modeling, combustion problems, two-phase flow, density driven flow and multi-component transport in porous media. Here we report on new developments for a parallel algebraic multigrid solver and applications to an eigenvalue solver, to flow in porous media and to an simulation of Navier-Stokes equations with turbulence modeling.

1 Introduction

In many cases the modeling of physical and technical problems leads to a description by partial differential equations. Due to the complexity of the equations and the geometry involved often the mathematical problem can be treated only by numerical methods. Appropriate discretizations lead to large systems of (non-linear) equations to be solved. To make the numerical treatment feasable, advanced numerical methods in conjunction with High Performance Computing have to be applied. Unstructured grids, adaptivity, multigrid methods, and parallelism have proven to be an efficient approach. Unstructured grids are required by the complex geometries. Adaptivity has to be used to reduce the computational cost especially for three-dimensional simulations. Finally the use of both multigrid methods and High Performance Computing on massively parallel MIMD machines is indispensable to reduce the computational time.

To bring together the features mentioned above the program package *UG*[1] has been designed during the last decade. It provides a platform for the discretization and the solution of partial differential equations on parallel computers.

The effort for building up a new application (in lines of code) on top of the UG platform is illustrated in figure1. The shown example is two-phase flow in porous media. The coding of the problem specific program part (pm) has only the extend of about 12% of the total amount. All other parts are problem independent and may be used for other problem class implementations as well. The used symbols have the following meaning:

- pm: two-phase flow in porous media
- ug: kernel library for (multi)grid management, numerical procedures and graphics,
- ddd: the parallel programming model (dynamic distributed data),

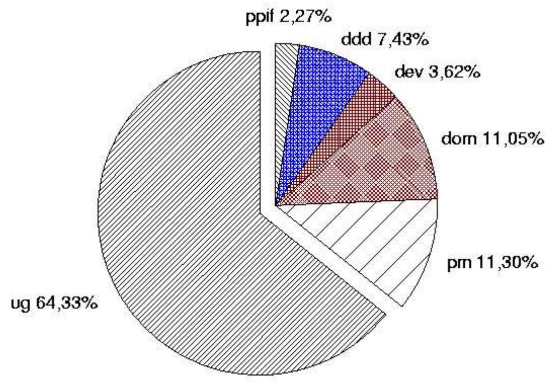

Figure 1. Effort to implement a parallel adaptive application in percent of lines of code (LOC).

- ppif: small and fast message passing interface (to shared memory on Cray T3E),
- dom: handling of the domain geometry,
- dev: devices drivers for X11, postscript and other devices.

This shows clearly the advantage of code reuse, when proper abstractions are introduced. Especially in the case of parallel adaptive applications a huge amount of work (and therefore development time) can be saved through the platform design. Furthermore the scientists involved in the problem class development do not need to treat detailed problems of parallel implementation, since only few lines of code have to be added to a problem class to get it running in parallel.

The kernel of *UG* has now reached a mature state and state-of-the-art numerical methods can be applied to complex and real-world problems. Applications to the Navier-Stokes equation with turbulence modeling, to combustion problems, to two-phase flow, density driven flow and multi-component transport in porous media and to flow in fractured rock have been realized, see also[2,3].

Nevertheless further work has to be done in various directions, as to mention

- improvement of dynamical load balancing[4],
- development of new parallel numerical algorithms,
- implementation of new applications.
- improvement of stability and efficiency

In this paper we report on some of these aspects. Section 2 reports to an porous media application with special emphasis on parallel performance. In Section 3 recent developments of the parallel algebraic multigrid solver[3] are discussed. Section 4 is devoted to the parallel solving of eigenvalue and plasticity problems. Finally in Section 5 an application to the Navier-Stokes equations with turbulence modeling is given.

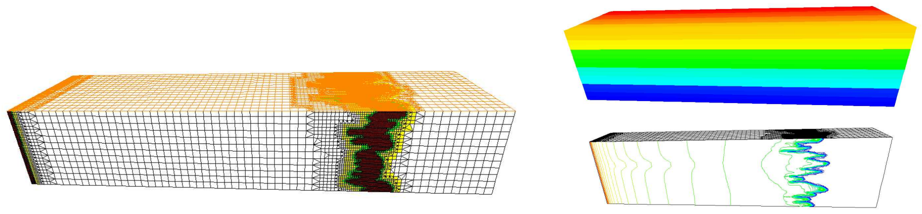

Figure 2. Adaptively refined multigrid with hexa/tetrahedra and pyramids; 16×16 stripped load balancing in flow direction; Vertical cutted domain with isolines of concentration.

PROCS	1	2	4	16	32	64	128	256
NLSOLVE [s]	233921	145842	72575	17597	9091	4599	2423	1354
$S_{NLSOLVE}$	1	1.60	3.22	13.3	25.7	50.9	96.6	173
ADAPT [s]	3413	2426	1541	691	502	347	314	271
S_{ADAPT}	1	1.41	2.21	4.94	6.80	9.84	10.9	12.6
TOTAL [s]	237394	148268	74116	18288	9593	4946	2737	1625
S_{TOTAL}	1	1.6	3.20	13.0	24.7	48.0	86.7	146

Table 1. Fixed speedup for two-phase flow: NLSOLVE (nonlinear solution in seconds), $S_{NLSOLVE}$ (speedup for this phase), ADAPT (grid adaption in seconds), S_{ADAPT} (speedup for this phase), TOTAL (overall time in seconds), S_{TOTAL} (overall speedup)

2 Two-Phase Flow

This section evaluates the speedup when a fixed sized problem should be computed fast for productivity reasons. Fast computation of a given problem is for example advantgeous, if parameter studies are necessary for optimization.

Two phase flow problems with phases consisting of liquids with different viscosity form fronts which tend to be instable. These instabilities can be seen in real live or physical experiments as fingers. These fingering results from a statistical behaviour, where interactions between the two phases on a very small scale influence the structure of the viscous front on a visible level.

As test scenario we choose a two-phase flow problem, where water displaces oil in a cubic channel (see figure 2). Interesting are the different fingering patterns evolving from different viscoscity parameters of the two phases. Since the resolution of the highly nonlinear effects at the front must be quite fine, the combination of parallelism and adaptivity tends to be an optimal approach. As load balancer a simple Recursive Orthogonal Bisection Method (RCB) is applied to distribute the coarse grid levels in flow direction. Thus important coupling information of the problem is kept processor local to avoid degradation of the solver's convergence rate. No load balancing need to be done during computation, since load remains equally distributed between the processors when the front moves.

Table 2 shows the speedup of the displacement example for the first 34 timesteps. The minimal amount of serial main memory needed is about 2 GB to allow the front to finger after about 100 timesteps. This corresponds to about 1 million unknowns which have to treated in each timestep. Results for one to eight processors are computed on a IBM SP-256, from 8 to 256 processors on Cray T3E at NIC. Timings are scaled by a constant factor, which is gained from the comparison of the 8 processor run on both machines.

The nonlinear solution process scales quite nice, giving a speedup of 173 for 256 processors. Speedup losses have two reasons: The first is the introduction of communication in the linear solver, when going from one to two processors, speedup loss is 20%. Second the convergence rates becomes with increasing processor count more worse. The cycle time of one linear iteration step scale good, which shows a stable scaling of the machine itself.

Grid adaption shows a scaling which is not comparable to that of the nonlinear solution. A speedup factor of 12.6 could be achieved in the 256 processor configuration. Analyzation of the losses has shown, that the grid manager itself has a better speedup behaviour, but the underlying programming modell DDD has a bottleneck when the interfaces for communication are rebuild after grid adaption. This process can be speeded up by reimplementation of the DDD interface module.

The total efficiency of the 256 processor run is 0.57, which can be treated as a quite satisfying overall result for a fixed sized problem.

3 Parallel Algebraic Multigrid FAMG

Besides classical (or geometric) multigrid methods purely algebraic multigrid methods (AMG) are desirable for many reasons; AMG considers only the stiffness matrix but not any geometric information such as elements or coordinates. AMG has the potential to solve complicated problems better than other methods. Even in a geometric multigrid AMG can be applied namely as the coarse grid solver. A third point of interest is the coupling of multigrid methods with already existing programs. The smallest possible interface—the matrix itself—is already feasible for AMG and the solver has no interaction with the geometry. A major drawback of many existing AMG methods is the missing or even impossible parallelization.

The starting point for our parallel AMG is the filtering AMG (FAMG) by WAGNER[5,6]. The main idea is to choose for each node good parent pairs of nodes for eventual elimination which ensures a certain filtering condition and leads to exactness on a given subspace. The best pairs are selected to eliminate the corresponding nodes; this parent nodes persist on the next coarser grid level and restriction and prolongation matrix entries with individually calculated values are installed between the nodes and their parents. The recursive application of this process yields a grid hierarchy on which standard multigrid algorithms can be realized.

For parallelizing this FAMG several additional steps have to be done. Since the elimination of a node influences its neighborhood, and thus the following selections for elimination, the cardinal point for the parallel FAMG will be to break up this sequential order in a suitable way. Due to the locality of the direct influence of a node our approach will divide the nodes into two classes: those which are influenced directly by nodes on other processors and the rest. Whereas the latter can be eliminated locally on each processor, the processing of the first ones needs a partly synchronization. Therefore a parallel graph coloring method is used[7]. For further details see[3].

Now we present first results of the new parallel FAMG. We examine the Laplace operator on the unit square with a tensor product mesh (called *structured* case) and on the shape of Lake St. Wolfgang (Austria) with an *unstructured* mesh from a grid generator. The structured mesh is distributed by a special load balancer to achieve tensor product like processor configurations, the unstructured mesh is distributed by recursive coordinate bisection (RCB). For anisotropic model problems the equation $\epsilon u_{xx} + u_{yy} = f$ is solved with the anisotropy parameter ϵ.

Tab. 2 shows that the convergence rate is rather independent from the mesh width respectively the number of nodes. This is essential to solve large problems efficiently.

	structured							unstructured		
	isotropic			anisotropic 10^{-6}						
unknowns	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]	unknowns	conv. rate	nr. it.	time [sec]
16641	0.053	7	6.6	10^{-5}	2	4.2	14095	0.057	7	4.6
66049	0.054	7	9.7	0.001	3	6.3	55637	0.123	9	9.4
263169	0.054	7	17.3	0.055	7	13.2	221065	0.206	12	20.7
1050625	0.057	7	38.1	0.044	6	31.8	881297	0.246	14	51.7
4198401	0.064	7	110.1	0.041	6	78.0	3519265	0.256	14	137.2
16785409	0.068	7	295.3	0.036	6	233.9	14065217	0.258	14	391.6

Table 2. The convergence rate depends only weakly upon mesh width (structured grid on a 16x8 processor configuration, unstructured grid on 128 processors).

Fig. 3 shows that the solver is very robust with respect to the variation of the anisotropy parameter. The convergence rate is bounded by 0.15 even for 128 processors and the solution time is quite independent of this parameter; this holds from low load up to full load examples.

Next we evaluate the quality of the parallelization. We consider the speedup where the same problem is solved on different number of processors. Tab. 3 indicates a quite constant convergence rate and the solution time decreases nearly by the factor as the number of processors increases.

We have seen very promising features of the new parallel FAMG. Nevertheless, further model problems (e. g. jumping coefficients and convection) should be inspected to explore

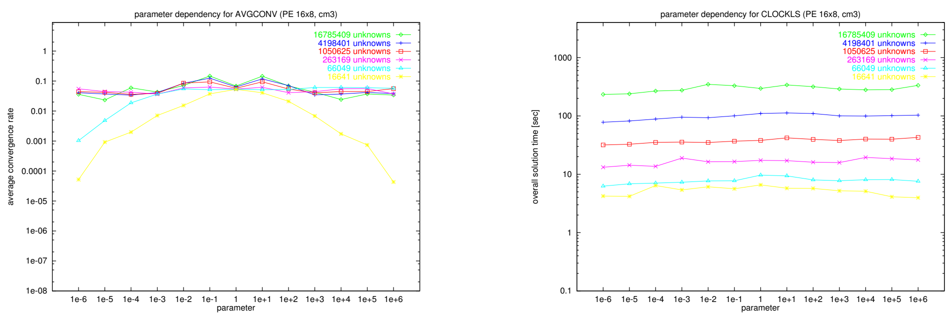

Figure 3. Robustness against anisotropy (128 processors, different loads).

the benefits and the limitations of the new method. At the end real world problems should be solved.

	structured						unstructured		
	isotropic			anisotropic 10^{-6}					
PE	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]	conv. rate	nr. it.	time [sec]
8	0.058	7	220.6	10^{-4}	2	140.7	0.237	13	235.8
16	0.067	7	135.8	0.053	7	123.2	0.194	12	179.6
32	0.062	7	81.4	0.046	6	70.7	0.241	13	97.6
64	0.059	7	58.3	0.043	6	43.7	0.240	13	78.4
128	0.057	7	38.1	0.044	6	31.8	0.246	14	51.7

Table 3. Speedup on different processor configurations (structured grid with 1 million unknowns, unstructured grid with 880000 unknowns)

4 A Flexible Finite Element Library

In *UG*, a general finite element library is included[8] which supports various discretizations, and which is especially adapted to problems in nonlinear solid mechanics[9]. Here, the applicability and flexibility of the finite element module is illustrated on the following model problem in eigenvalue computations.

Let $\Omega \subset \mathbf{R}^d$, $d = 2, 3$ be a domain. We solve the eigenvalue problem:
find $(w_i, \lambda_i) \in H^1(\Omega) \times \mathbf{R}$ such that

$$\int_\Omega \nabla w_i \cdot \nabla v \, dx = \lambda_i \int_\Omega w_i v \, dx, \qquad v \in H^1(\Omega).$$

This corresponds to the eigenvalue problem

$$-\Delta w_i = \lambda_i w_i \text{ in } \Omega, \qquad w_i \cdot n = 0 \text{ on } \partial\Omega$$

in the strong formulation; in particular, we impose Neumann boundary conditions. Thus, we know a priori the first trivial eigenpair $w_0 \equiv 1$ and $\lambda_0 = 0$. Note that this results in a singular stiffness matrix (which, of course, has severe consequences for the solution process).

The eigenmode approximations are computed by a block inverse iteration with a full Ritz-Galerkin orthogonalization in every step. Starting from initial guesses $\tilde{w}_1, ..., \tilde{w}_m$, we perform the following algorithm:

a) Ritz-Galerkin step: We form the small matrices

$$M = \left(\int_\Omega \nabla \tilde{w}_i \cdot \nabla \tilde{w}_j \, dx \right), \qquad N = \left(\int_\Omega \tilde{w}_i \tilde{w}_j \, dx \right),$$

and we solve the generalized eigenvalue problem

$$Mx_i = \lambda_i N x_i, \qquad x_i \in \mathbf{R}^m;$$

then, we define approximations by

$$w_i = \frac{1}{x_i^T N x_i} \sum_{j=1}^{m} x_{ij}\, \tilde{w}_j, \qquad i = 1, ..., m.$$

b) Block inverse iteration step: We update the space of test functions by solving

$$\int_\Omega \nabla \tilde{w}_i \cdot \nabla v\, dx = \lambda_i \int_\Omega w_i v\, dx, \qquad v \in H^1(\Omega), i = 1, ..., m$$

and return to a).

For the solution of the linear problems, we use a fixed number of parallel multiplicative multigrid cycles with Krylov acceleration and Gauß-Seidel smoother, where we project in every step the solution onto the space which is orthogonal to w_0.

As an example, we present the parallel results of the eigenmode computations on a domain representing the surface of a large lake (Bodensee in Germany). The shape of the eigenmode w_{11} is visualized in Fig. 4 (using *GRAPE*[10]), the first 4 of 16 computed eigenvalues are listed in Tab 4.

discretization	P_1	P_2	P_2	P_2
elements	1620992	101312	405248	1620992
unknowns	816737	205745	816737	3254465
λ_0	0.00000e-00	0.00000e-00	0.00000e-00	0.00000e-00
λ_1	8.88113e-04	8.88274e-04	8.89473e-04	8.88088e-04
λ_2	2.46599e-03	2.46597e-03	2.46582e-03	2.46576e-03
λ_3	4.77479e-03	4.77490e-03	4.77461e-03	4.77447e-03

Table 4. Comparison of the numerical solution with different discretizations and different mesh sizes of the 4 lowest eigenvalues of the Bodensee.

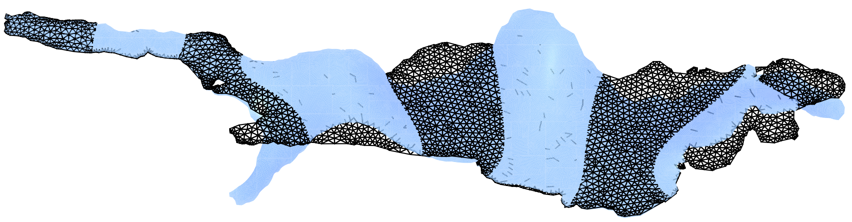

Figure 4. The 11th eigenmode w_{11} of the Bodensee.

level	2	3	4	5
elements	101312	405248	1620992	6483968
unknowns	205745	816737	3254465	12992897
32 processors	21 min.	49 min.		
	(21 sec.)	(50 sec.)	(190 sec.)	
64 processors	11 min.	28 min.	93 min.	
	(13 sec.)	(26 sec.)	(99 sec.)	
128 processors	7 min.	15 min.	47 min.	
	(9 sec.)	(14 sec.)	(44 sec.)	(169.8 sec.)

Table 5. Parallel performance of the eigenvalue computation (average time for the linear solver in every step) for P_2-elements. Due to the cpu-time-limit, not sufficiently many steps of the block inverse iteration could be performed on level 4 (32 proc.) and level 5 (128 proc.).

The parallel performance (see Tab. 5) underlines the efficiency of the method, which is asymptotically of optimal complexity and optimal parallel scalability (due to the large coarse grid problem with 6332 elements we obtain optimal multigrid efficiency not in the first refinement step). Since we use a second order discretization and millions of unknowns, the results are reliable due the monotone asymptotic convergence; by simple extrapolation, we expect an accuracy of at least 0.1%.

Nevertheless, fully reliable results can be obtained only by the computation of rigorous eigenvalue inclusions; this is realized using interval arithmetic, and first results (on more simple geometries) are already documented in[11].

A second quite different application is a plasticity experiment on a gear geometry. The calculation consists of 27 load steps and is performed using 128 Processors of the Cray T3E-1200 at NIC, see figure 5. In the context of the UG platform this example demonstrates the full functionality included in the software: multigrid consisting of tetrahedra, prims and hexahedra, grid adaption, and dynamic load balancing. This features are combined with a nonlinear solver using a multiplicative multigrid scheme and a discretization based on Prandl-Reuss Plasticity. Since this is a recently performed calculation no detailed timings have been done yet. Next steps will be the analyzation of speedup and scalability behaviour.

5 Navier-Stokes Equations and Turbulence Modelling

The Navier-Stokes library in *UG* uses a Finite Element based Finite Volume Method with colocated variables. Since a colocated scheme is not stable, a special stabilization scheme is applied which introduces a physical advection correction scheme to couple the momentum equation and the pressure equation. This results in a Laplacian term for pressure in the continuity equation scaled with the mesh size squared and therefore tends to zero as the grid is refined. This physical advection correction scheme called FIELDS was developed by Raw[12]. The idea is to solve in each element a Finite Difference approximation of the linearized momentum equation at all integration points. The resulting integration

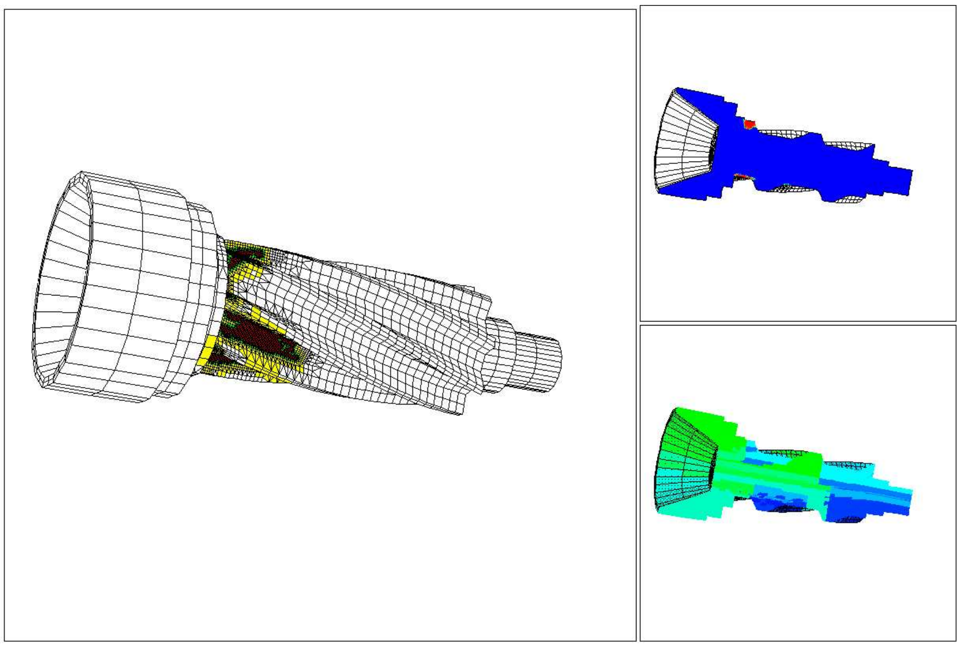

Figure 5. Plastic behaviour of a gear under load. Left picture shows grid adaption, right upper picture shows plastic zones, right lower visualizes the load balancing of the geometry onto 128 processors

point velocities depend on all corner values of velocity and pressure. After insertion in the continuity equation a pressure dependence and full coupling of all equations is gained. This can be done independently at all elements and is therefore very advantageous for parallelization.

A special problem class in the Navier-Stokes community is turbulence modelling. A very promising way to simulate turbulent flow charactetristics is the so called Large Eddy Simulation (LES). In contrast to Reynolds averaged turbulence models (RANS), it is not based on time averages but on local volume averages. This means that large structures have to be resolved and only the small ones are modelled. In RANS methods all structures are modelled with the difficulty that a suitable turbulence model is not easy to design. LES models use the fact that small structures are nearly isotropic and more universal than large structures and therefore modelling becomes simpler. But to reach the necessary resolution very fine meshes have to be used and because of this parallel calculations are needed.

The averaged equations for LES are derived by applying filter operators (for example a volume-average box filter) to the governing equations. For the momentum equation this results in:

$$\frac{\partial \overline{u_i}}{\partial t} + \frac{\partial}{\partial x_j}\left(\overline{u_i}\,\overline{u_j}\right) + \frac{\partial \overline{p}}{\partial x_i} - \frac{\partial}{\partial x_j}\left(\nu\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right)\right) + \frac{\partial}{\partial x_j}\tau_{ij} = 0$$

where an overbar denotes an average value. The subgrid scale stress tensor τ_{ij} is modelled

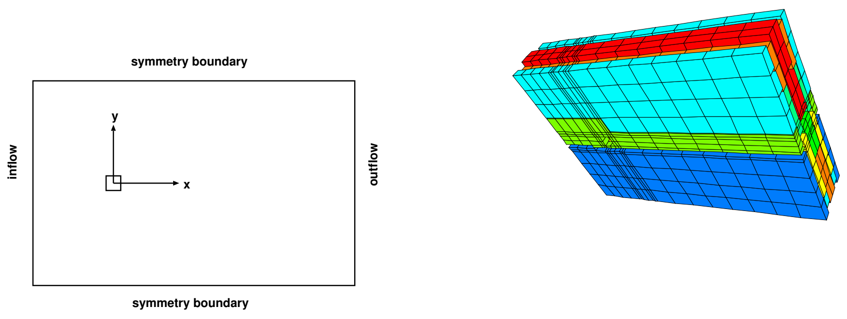

Figure 6. Sketch of the domain and load balance on 8 processors

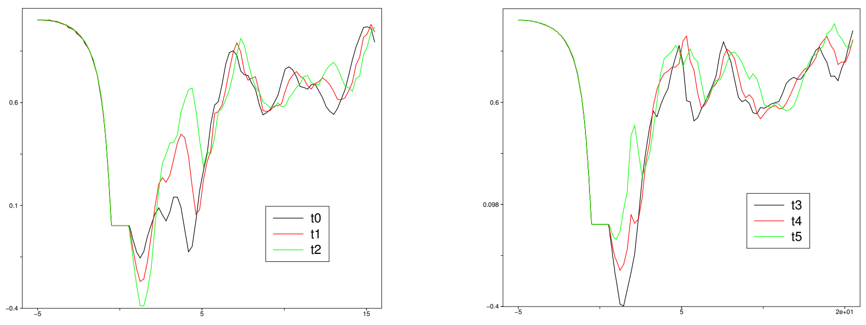

Figure 7. u-component of the velocity vector

by the dynamic model of Germano[13]. This model, in contrast to the Smagorinsky model[14], for which a constant and universal model parameter is assumed, is truly local and hence it is able to reflect approximately local flow phenomena. Finally the model has the form:

$$\tau_{ij} - \frac{1}{3}\delta_{ij}\tau_{kk} = -2C\Delta^2|\overline{S}|\overline{S_{ij}}$$

with $\overline{S_{ij}} = \frac{1}{2}\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right)$ and $|\overline{S}| = \sqrt{2\overline{S_{ij}}}$.

An interesting problem is the flow around an infinite long square cylinder as described in the workshop of Rodi et al.[15]. The infinite dimension in axis direction can be numerically realized using periodic boundary conditions since the flow structures in that direction can be assumed to be almost periodic. A sketch of the domain is given in Fig. 6. For implementing periodic boundary conditions in *UG* each element at the periodic boundary and its partner at the opposite side have to be assigned to the same processor. For this reason the load balancing strategy RCB of CHACO has been modified to handle periodicity. An example for 8 processors can be seen in Fig. 6.

The Reynoldsnumber for this problem was 1000. The grid is built of 800000 hexahedrons which correspond to 3 million unknowns. The nonlinear system is linearized by a Quasi-Newton method and the resulting linear system is solved by the Kryloc subspace method BiCGSTAB with multigrid as preconditioner. As smoother the incomplete LU factorization with β modification was employed and the time solver was a diagonally implicit Runge-Kutta-method of second order. In Fig. 7 the solution of the velocity component in x-direction on a line through the midpoint of the cylinder in mean flow direction is shown at 6 successive points of time illustrating the complex behaviour of the flow.

6 Conclusions

With the program package *UG* we follow the strategy of combining advanced numerical methods with the advantages of high-performance computers like a Cray T3E. It has been shown that on the base of the parallel *UG* platform both the development of new, highly efficient numerical algorithms and the solution of complex problems is feasable. The obtained results show clearly that

- the computational time can be reduced significantly,
- for some problems parallel computers are neccessary, due the limitations of memory size on sequential machines,
- High Performance Computing becomes an enourmous factor of productivity, when treating real world problems.

Therefore the availability of HPC computing power will have a strong impact on the research related to practical applications of scientific and industrial importance.

Acknowledgments

The benchmark computations were performed with computer time provided by HLRZ in Jülich and HLRS in Stuttgart.

References

1. P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert, and C. Wieners. UG - a flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science*, 1:27–40, 1997.
2. P. Bastian, K. Birken, K. Johannsen, S. Lang, V. Reichenberger, C. Wieners, G. Wittum, and C. Wrobel. A parallel software-platform for solving problems of partial differential equations using unstructured grids and adaptive multigrid methods. In E. Krause and W. Jäger, editors, *High performance computing in science and engineering*, pages 326–339. Springer, 1999.
3. P. Bastian, K. Birken, K. Johannsen, S. Lang, V. Reichenberger, C. Wieners, G. Wittum, and C. Wrobel. Parallel solutions of partial differential equations with adaptive multigrid methods on unstructured grids. In *High performance computing in science and engineering II*, 1999. submitted.
4. S. Lang. *Parallele Numerische Simulation instationärer Probleme mit adaptiven Methoden auf unstrukturierten Gittern*. PhD thesis, Universität Heidelberg, 2001.

5. Christian Wagner. On the algebraic construction of multilevel transfer operators. *Computing*, 2000, to appear.
6. R. E. Bank and C. Wagner. Multilevel ILU decomposition. *Numerische Mathematik*, 82:543–576, 1999.
7. R. K. Gjertsen Jr., M. T. Jones, and P. E. Plassmann. Parallel heuristics for improved, balanced graph colorings. *Journal of Parallel and Distributed Computing*, to appear.
8. C. Wieners. The implementation of adaptive multigrid methods for finite elements. Technical report, Universität Stuttgart, SFB 404 Preprint 97/12, 1997.
9. C. Wieners. Multigrid methods for Prandtl-Reuß-plasticity. *Numer. Lin. Alg. Appl.*, 6:457–478, 1999.
10. M. Rumpf and A. Wierse. Grape, eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forschung und Entwicklung*, 7:145–151, 1992.
11. H. Behnke, U. Mertins, M. Plum, and C. Wieners. Eigenvalue inclusions via domain decomposition. 1999. submitted.
12. G. E. Schneider and M. J. Raw. Control volume finite-element method for heat transfer and fluid flow using colocated variables. *Numerical Heat Transfer*, 11:363–390, 1987.
13. M. Germano. Turbulence: the filtering approach. *Journal of Fluid Mechanics*, 238:325–336, 1992.
14. J. Smagorinsky. General circulation experiments with the primitive equations i. the basic experiment. *Monthly Weather Review*, 91:99–164, 1963.
15. W. Rodi, J. H. Ferziger, M. Breuer, and M. Pourquié. Status of large eddy simulation: Results of a workshop. *Journal of Fluids Engineering*, 119:248–262, 1997.